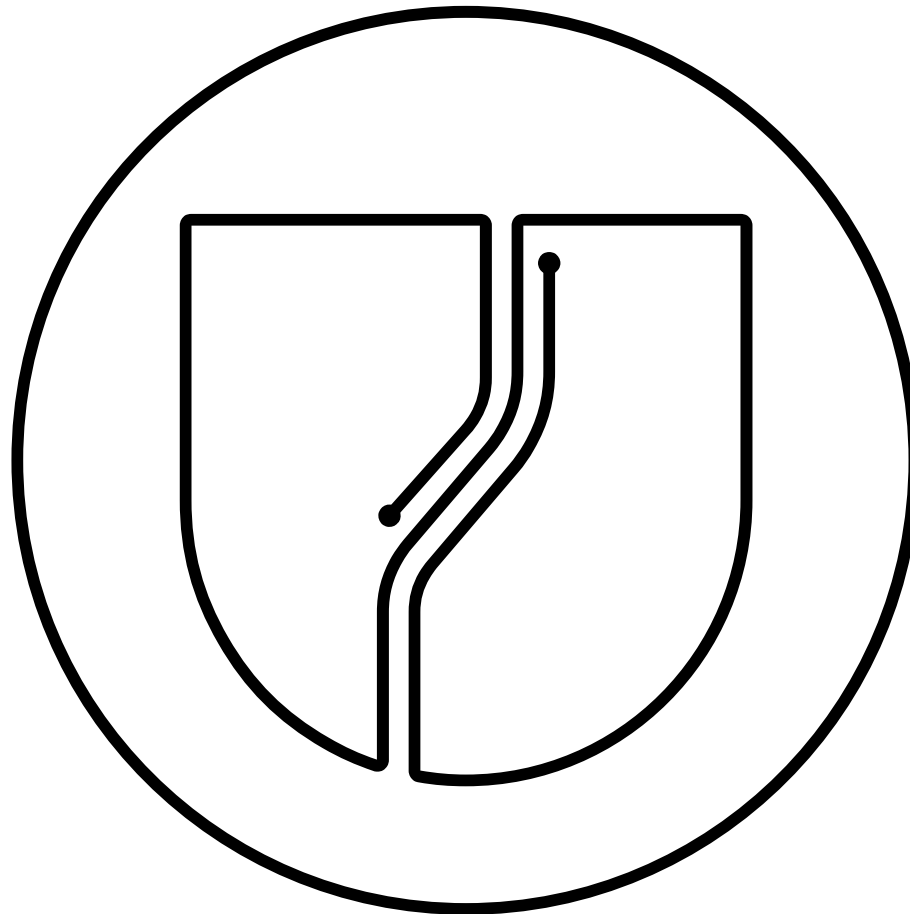


## Updating an Embedded System



# About me

We update

- Me:
  - Software Engineer at DENX, Gmbh
  - U-Boot Custodian for NXP's i.MX
  - Focus on Linux embedded
  - Author of FOSS SWUpdate

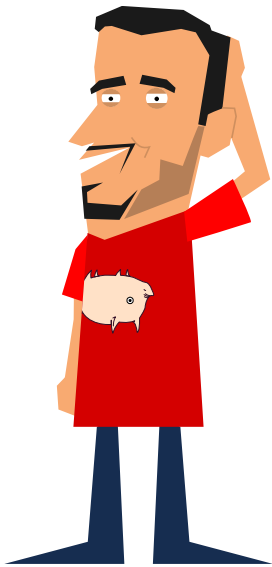
# Do we update ?

We update



# Local Update

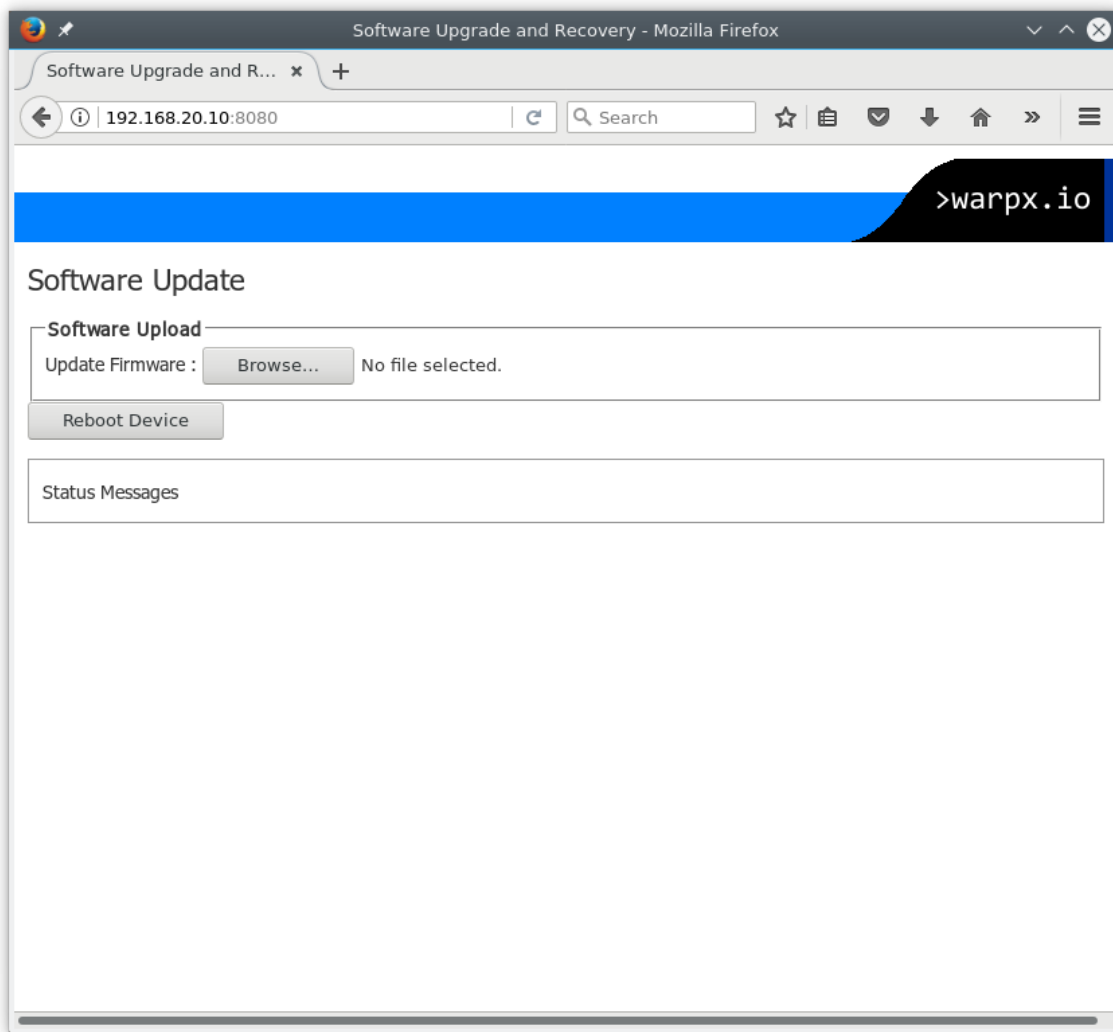
We update





# Push Software

We update

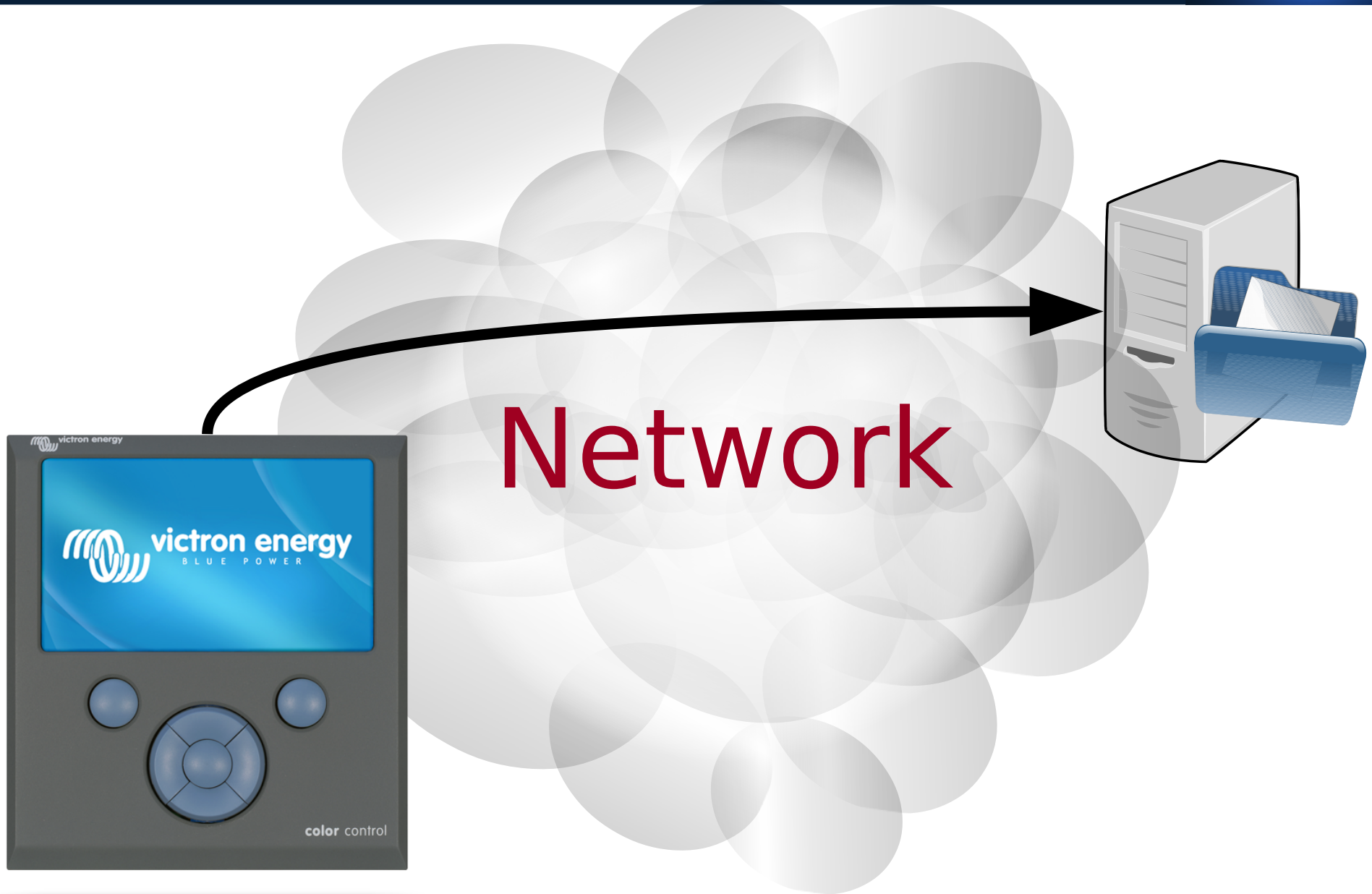


Network



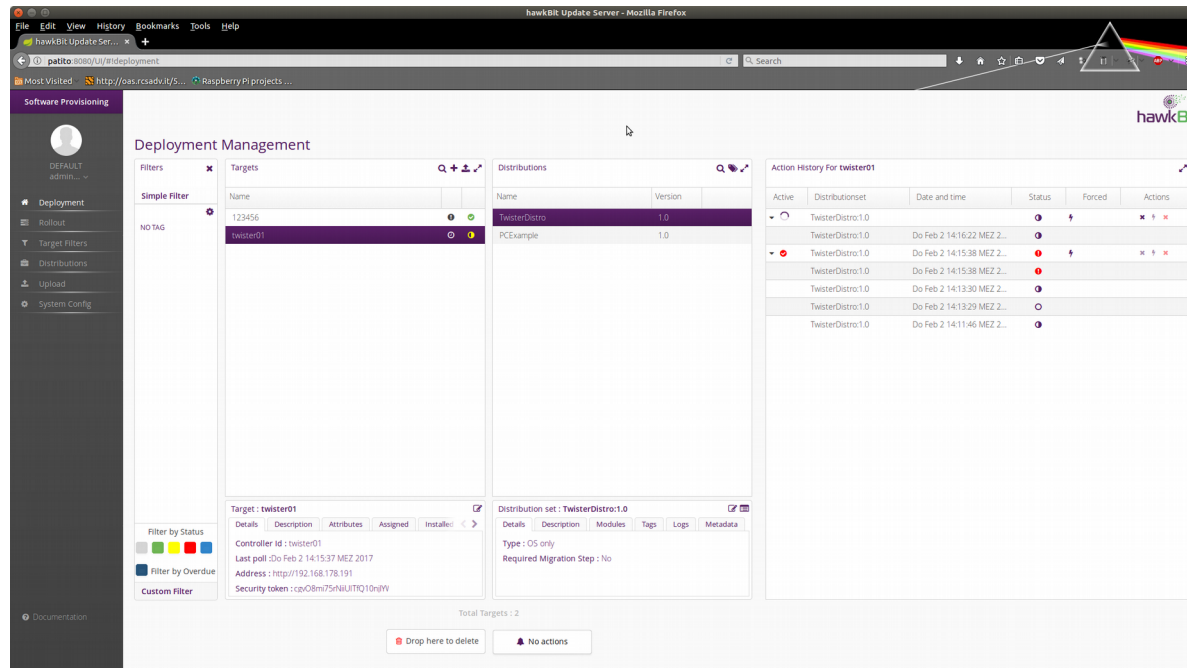
# Pull Software

We update



# Deployment systems

We update



Network



# Rescue system

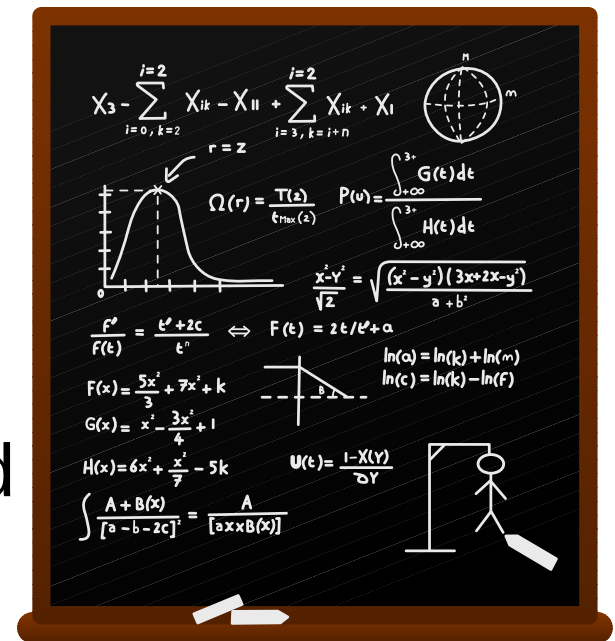
We update



# Requirement of updater ES

We update

- Power-off safe
- Must not brick the device
- Atomic: must not apply half an update
- Secure
  - Signing images and verification of images
  - Prevent that device can be hijacked



# Requirements - 2

We update

- Remote unattended update
- Update of bootloader, kernel, filesystem
- Failsafe, Apply / rollback system updates
- It should take care of most important law

## Murphy's Law

If it can go wrong,  
it will go wrong.

As much as possible !

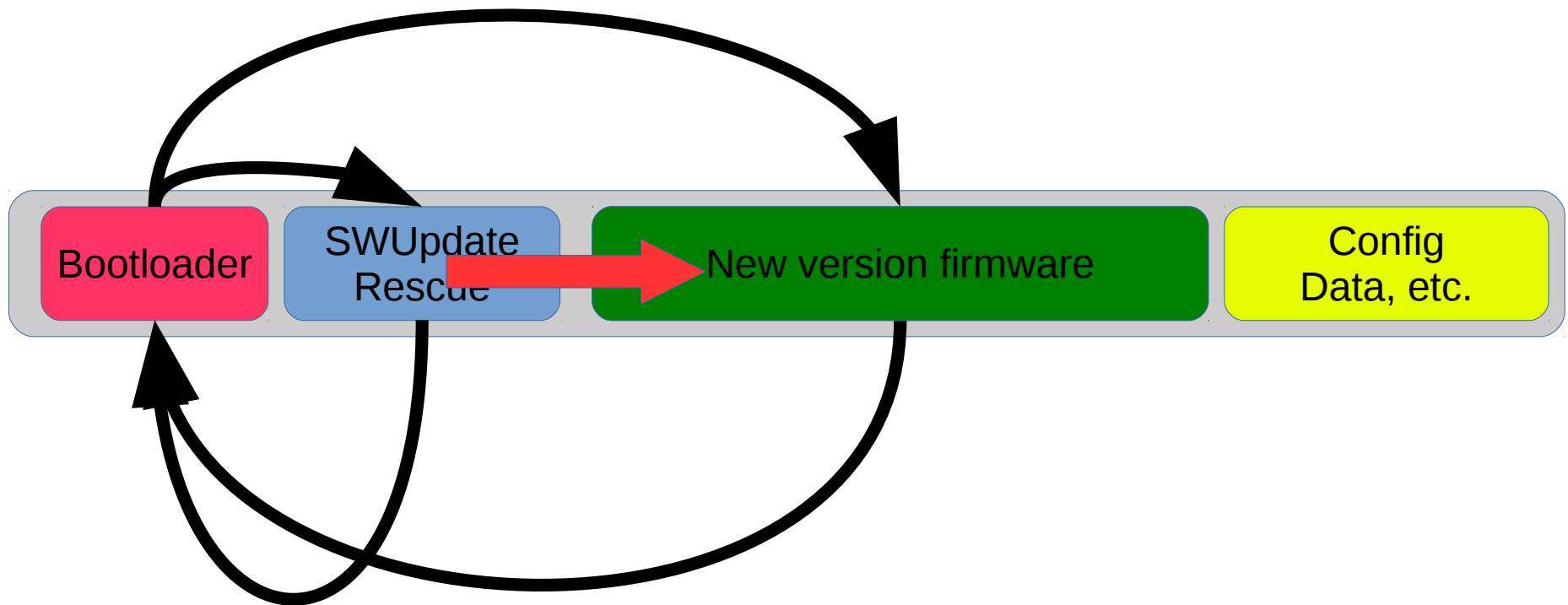
# Components to be update

We update

- Bootloader
- Kernel
- Root filesystem
- System Application
- FPGAs
- Microcontroller, etc.
- Configuration

# Single copy -1

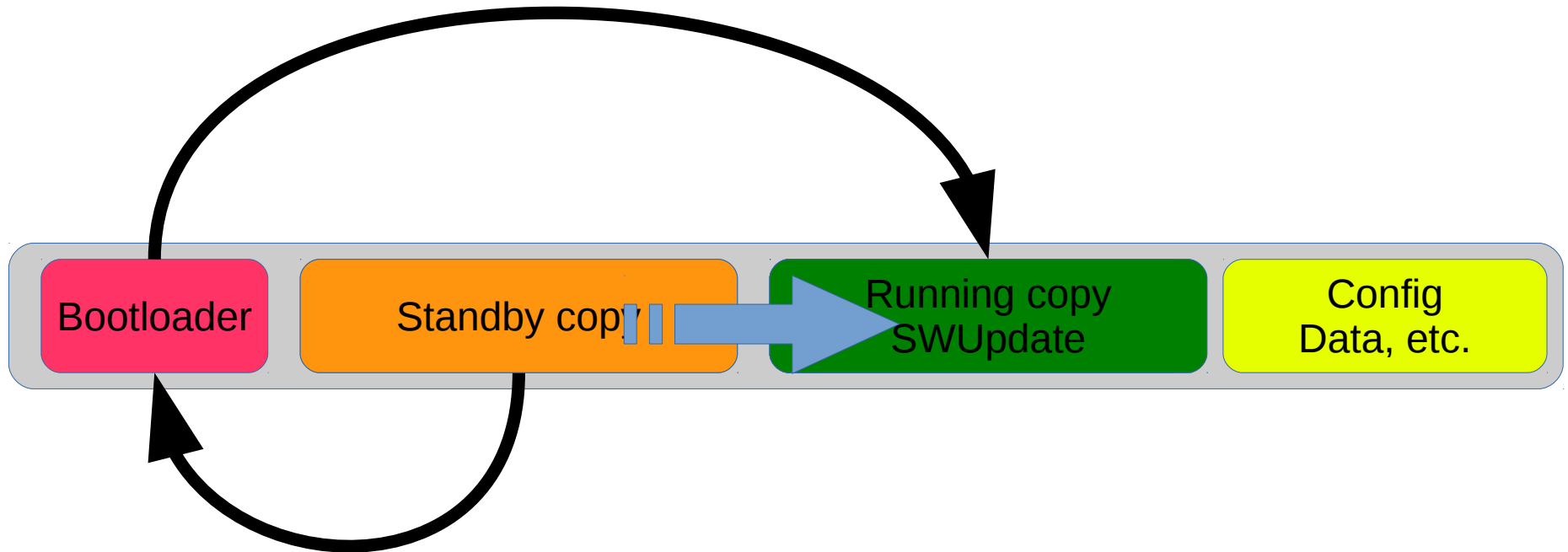
We update





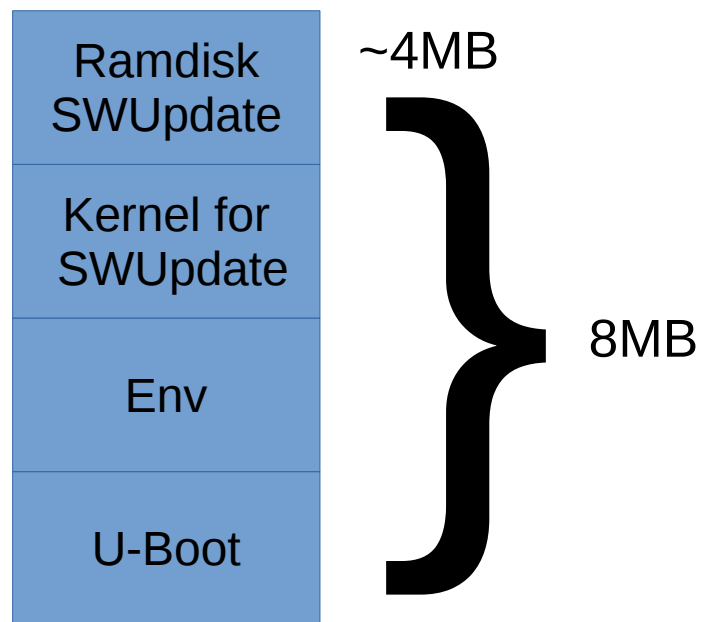
# Double-copy

We update

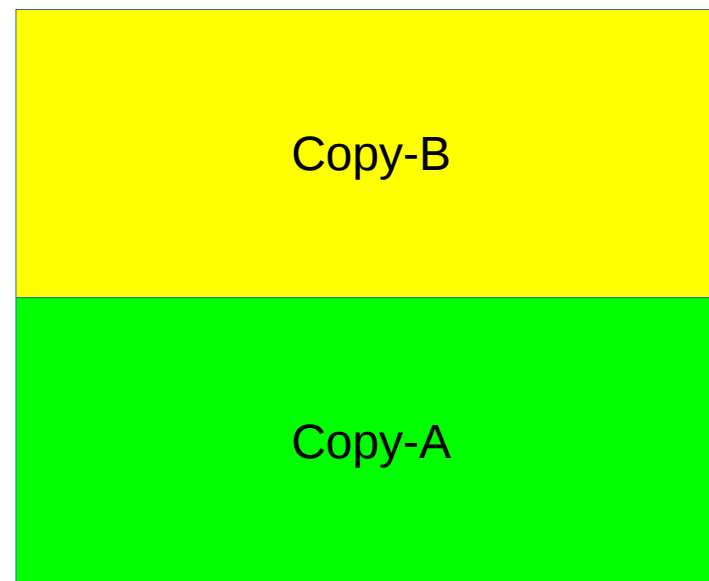


# Combine methods

We update



SPI NOR Flash



NAND / eMMC / SD

# SWUpdate



We update

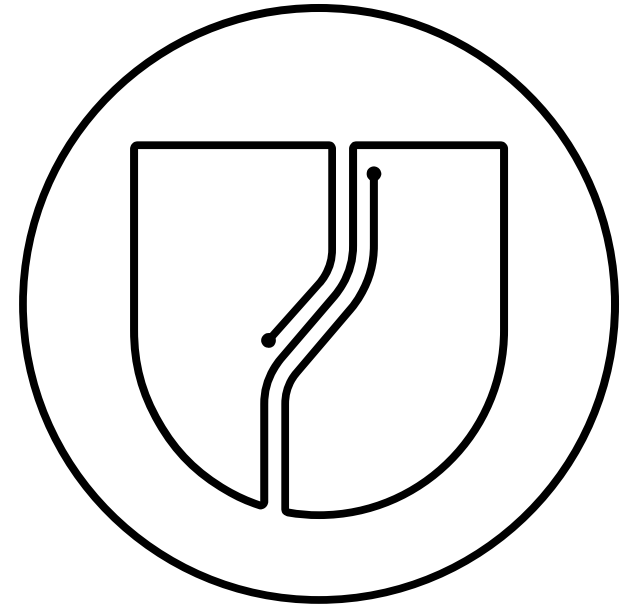
- Project started end 2014
- GPLv2, client library LGPLv2
- Often delivered together with BSP
- In the meantime:
  - ~40 developers sent contribution
  - Release cycle 3 months
  - One of Yocto updater:
    - [https://wiki.yoctoproject.org/wiki/System\\_Update](https://wiki.yoctoproject.org/wiki/System_Update)
  - Used by many devices in field

Deeds, not words !

# Features - Basis

We update

- Atomic update
- Embedded media
  - eMMC, SD
  - Raw NAND, UBI, NOR, SPI-NOR
- Single image (SWU) for multiple devices
- Power-Off safe
- Hardware-Software check

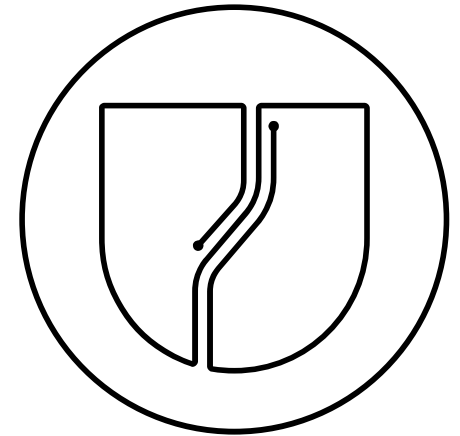


Deeds, not words !

# Features - Interfaces

We update

- Local Interface
- Remote interface / OTA
  - integrated web server (PUSH mode)
  - Backend: integrated REST client connector to hawkBit (PULL Mode)
  - remote server download (PULL Mode)
  - Custom interface (client library, LGPL)

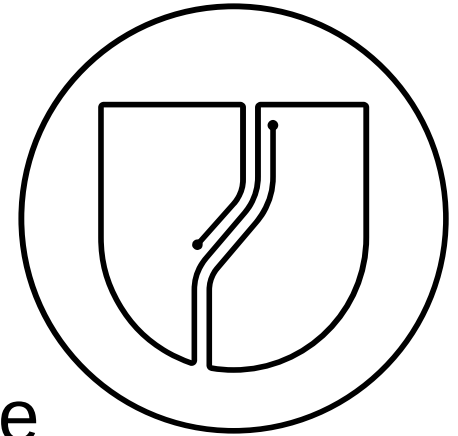


Deeds, not words !

# Features - Extended

We update

- Integrated LUA interpreter
  - modular with plugins in LUA
- Embedded Buildsystems
  - Integrated in Yocto with meta-swupdate
  - Officially supported by Buildroot
- Support for bootloader
  - U-Boot
  - GRUB
- Small footprint

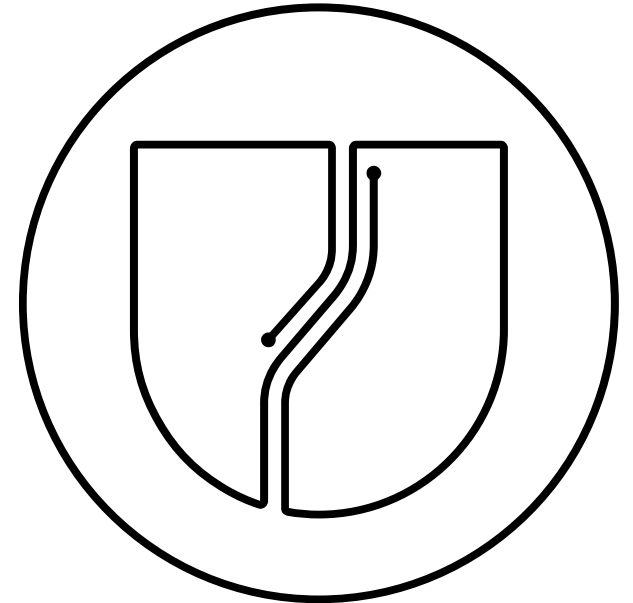


Deeds, not words !

# Features - next

We update

- Fallback with bootloaders
- Image updater and file updater
- Interface to report progress
- Uses Kbuild for configuration
- Streaming without temporary copies

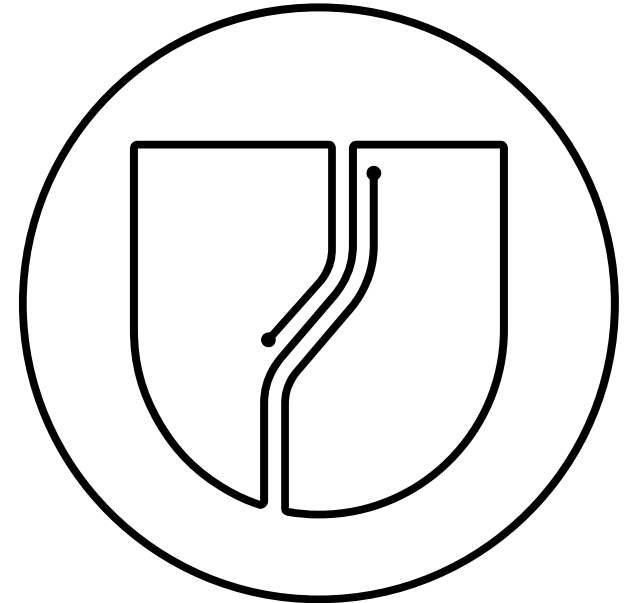


Deeds, not words !

# Features - Security

We update

- HTTPS protocol
- Use Certificates for server verification
- Signed Images
- Encrypted artifacts
- Privilege separation
  - Installer usually runs as root
  - Network processes runs on different user



Deeds, not words !



# Structure SWU image

We update

- CPIO format for simplicity
- sw-description describes update
- Images data / artifacts



# sw-description

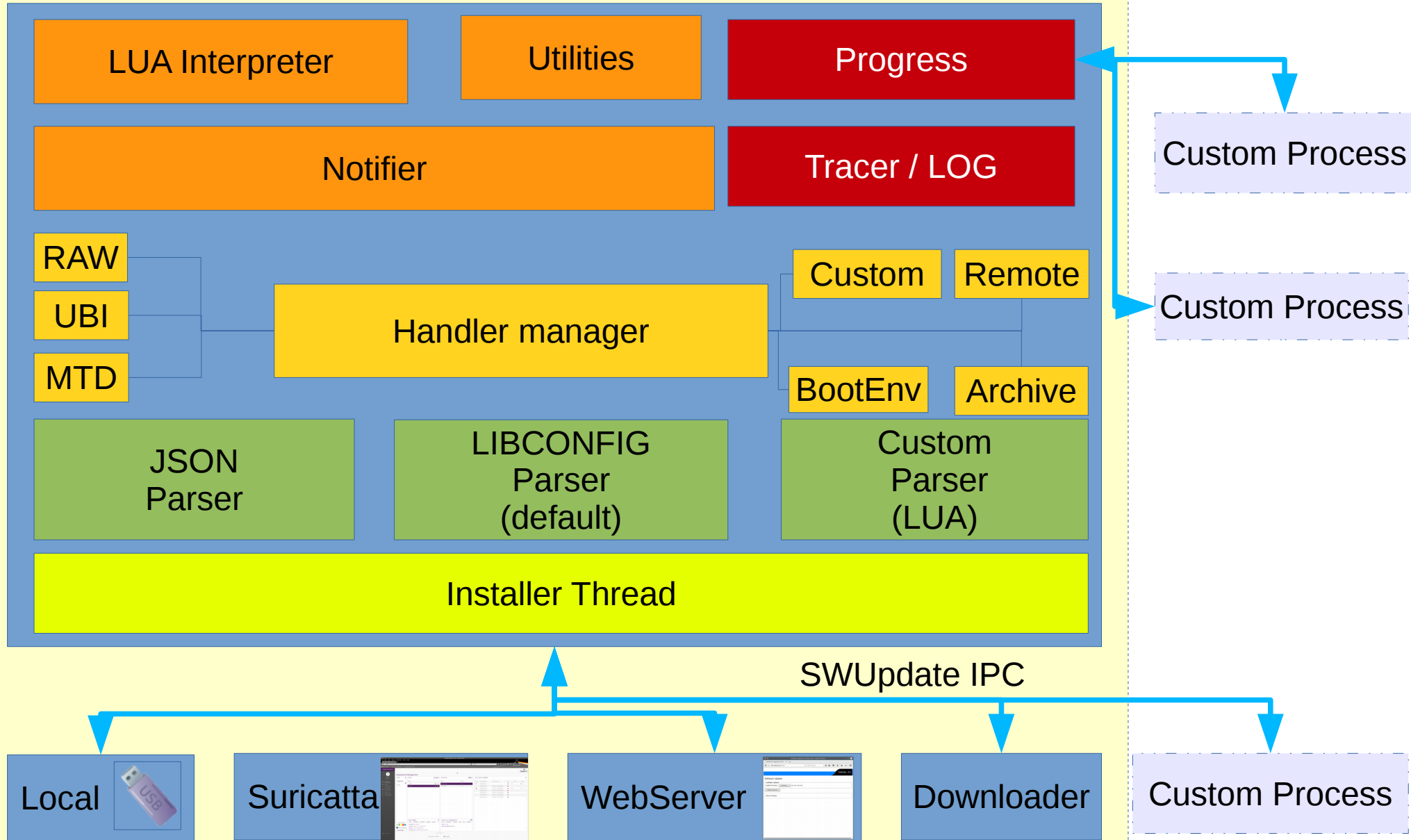
We update

- Describe how to install a release
- Different parser
  - libconfig (default)
  - JSON
  - Custom (LUA)
    - Example: XML parser using LUAExpat

# SWUpdate's architecture

We update

SWUpdate



# Config

We update

File Edit View Search Terminal Help

.config - Swupdate Configuration

## Swupdate Configuration

Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search.  
Legend: [\*] built-in [ ] excluded <M> module < > module capable

### Swupdate Settings --->

Bootloader (U-Boot, ..) (None) --->

[\*] Enable image downloading

[\*] Allow to add sha256 hash to each image

[ ] Enable verification of signed images

[ ] Images can be encrypted with a symmetric key

Suricata --->

[\*] Enable webserver

Webserver Features --->

Archival Features --->

Parser Features --->

Image Handlers --->

<Select>

< Exit >

< Help >

< Save >

< Load >

# Sw-description : structure

We update

```
Software = {  
  Version = "1.0.0";  
  
  myhw = {  
    hardware-compatibility : [ "1.0", "1.1", "1.3"];  
  
    images : (  
      {  
        filename = "rootfs.ext4.gz";  
        device = "/dev/mmcblk0p1";  
        type = "raw";  
      });  
  
    files : ({  
      filename = "archive.tgz";  
      type = "archive";  
      Path = "/usr/share/myapp";  
    });  
  
    scripts : (  
      {  
        filename = "postinstall.sh";  
        type = "shellscript";  
      });  
  }  
}
```

Header

Board specific

Section: images

Section: files

Section: scripts

# One image for multiple devices

We update

```
Software = {  
  Version = "1.0.0";
```

Header

```
  hmi = {  
    hardware-compatibility : [ "1.0", "1.1", "1.3"];
```

Target : HMI

```
    images : (  
      {  
        .....  
      });
```

```
  }
```

```
TypeA-1 = {  
  Hardware-compatibility : [ "2.1", "2.2", "3.3"];
```

Target: TypeA-1

```
  images : (  
    {  
      .....  
    });
```

```
}
```

```
}
```

# Collections

We update

```
software =
{
    version = "0.1.0";
    myhw = {
        hardware-compatibility: [ "1.0" ];
        stable : {
            copy1 : {
                images: (
                    {
                        filename = "core-image-full-cmdline-twister.ubifs";
                        type = "ubivol";
                        volume = "rootfs1";
                        sha256 = "@core-image-full-cmdline-twister.ubifs";
                    },
                    {
                        filename = "ulmage-twister.bin";
                        type = "flash";
                        device = "/dev/mtd10";
                        sha256 = "@ulmage-twister.bin";
                    }
                );
            };
            scripts: (
                {
                    filename = "test.lua";
                    type = "lua";
                    sha256 = "@test.lua";
                }
            );
            uboot: (
                {
                    name = "nandroot";
                    value = "rootfs1";
                },
                {
                    name = "kernelpart";
                    value = "kernel1";
                }
            )
        };
    };
};
```

# Collections

We update

```
copy2 : {
  images: (
    {
      filename = "core-image-full-cmdline-twister.ubifs";
      type = "ubivol";
      volume = "rootfs2"
      installed-directly = true;
      sha256 = "@core-image-full-cmdline-twister.ubifs";
    },
    {
      filename = "ulImage-twister.bin";
      type = "flash";
      device = "/dev/mtd11";
      sha256 = "@ulImage-twister.bin";
    }
  );
  scripts: (
    {
      filename = "test.lua";
      type = "lua";
      sha256 = "@test.lua";
    }
  );
  uboot: (
    {
      name = "nandroot";
      value = "rootfs2";
    },
    {
      name = "kernelpart";
      value = "kernel2";
    }
  );
};
```



# Handlers



We update

- flash devices in raw mode (both NOR and NAND)
- UBI volumes
- Archives (tarballs,...)
- raw devices, such as a SD Card partition
- U-Boot environment
- LUA scripts
- Shell scripts
- Remote handler

But you can also create your own ...

# Embedded Script

We update

- Executive part of sw-description
- Description changed at runtime
- Use cases for **Embedded Script**:
  - Check if an update is allowed
  - Set Partitions
  - Pre-install script



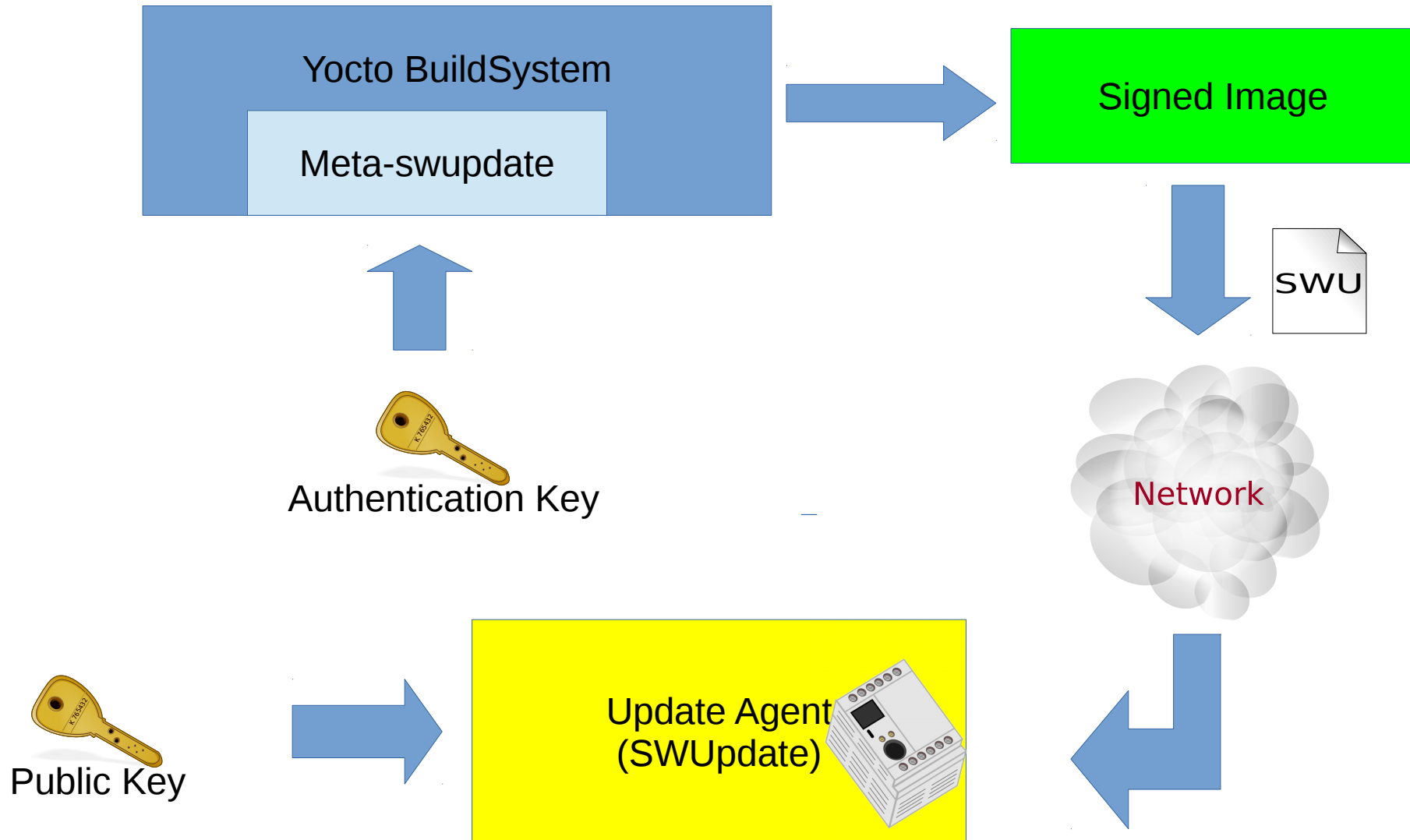
# Rollback

We update

- Together with U-Boot “bootcounter”
- Increment count in bootloader
- Reset after successful update / boot
- If reboots and count > threshold
  - Bootloader knows update / boot failed
  - Bootloader loads alternate boot

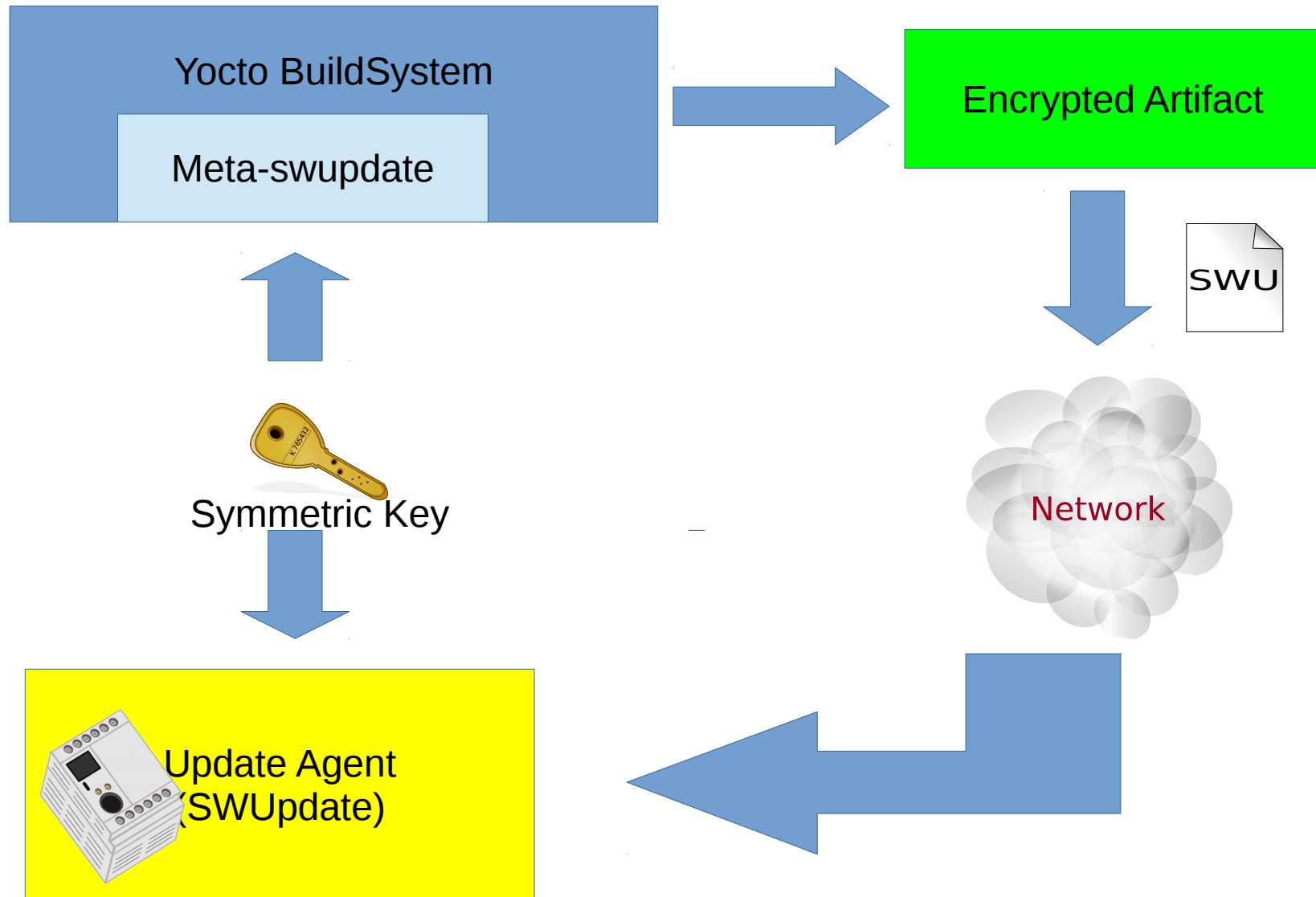
# Security: Signed images

We update



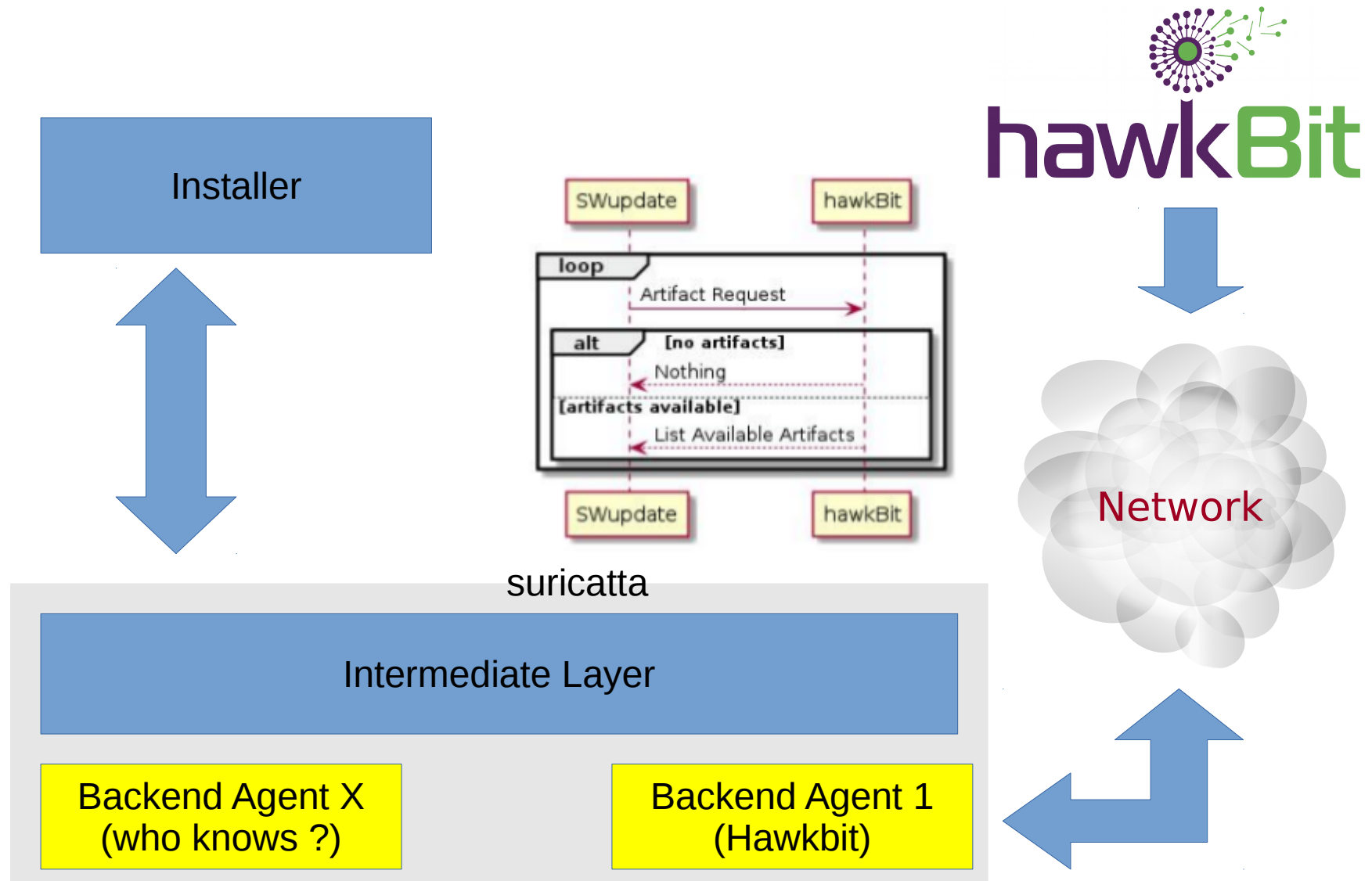
# Security: Encrypted images

We update



# Suricata mode

We update



# Automatic SWU Image build

We update



- meta-swupdate to build swupdate and swu
- Rescue image recipe
- Provides a class to automatically generate and sign a release image SWU

# Creating own SWU

We update

```
DESCRIPTION = "Example Compound image for beaglebone "  
SRC_URI_beaglebone = "file://sw-description \  
"
```

inherit swupdate

```
LICENSE = "MIT"  
LIC_FILES_CHKSUM = "file://${COREBASE}/LICENSE;md5=4d92cd373abda3937c2bc47fbc49d690 \  
file://${COREBASE}/meta/COPYING.MIT;md5=3da9cfbcb788c80a0384361b4de20420"
```

```
# IMAGE_DEPENDS: list of Yocto images that contains a root filesystem  
# it will be ensured they are built before creating swupdate image  
IMAGE_DEPENDS = ""
```

```
# SWUPDATE_IMAGES: list of images that will be part of the compound image  
# the list can have any binaries - images must be in the DEPLOY directory  
SWUPDATE_IMAGES = " \  
core-image-full-cmdline \  
"
```

```
# Images can have multiple formats - define which image must be  
# taken to be put in the compound image  
SWUPDATE_IMAGES_FSTYPES[core-image-full-cmdline] = ".ext3"
```

```
COMPATIBLE = "beaglebone"
```



# SWUpdate Roadmap

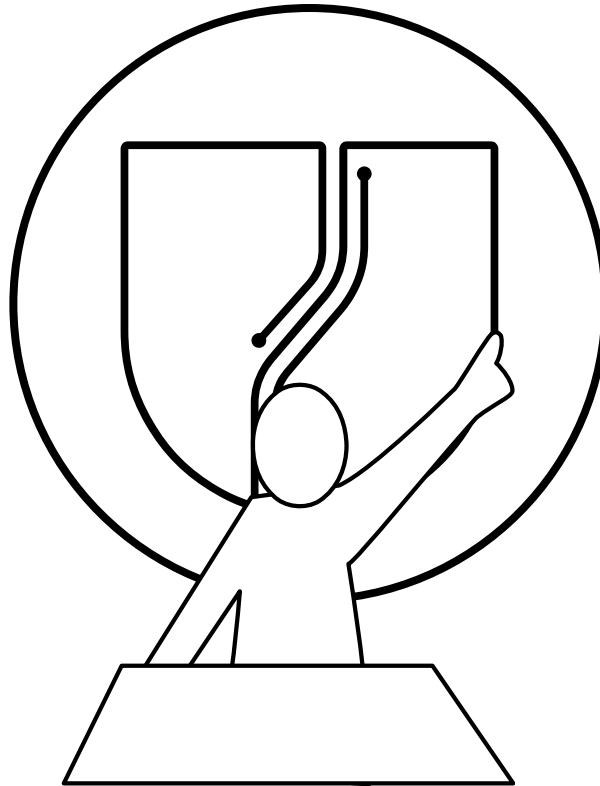


We update

- Extend community
- SWUpdate as Updater Gateway
- Dynamic LUA Handlers / new Handlers
- Hardware Keys / TPM for decryption
- Delta update
- Chain Handlers for single artifact
- Add other backends, support multiple servers
- A new modern Website

# Questions

We update



<http://sbabic.github.io/swupdate/>  
swupdate@googlegroups.com